

autogluon_each_location

October 9, 2023

```
[1]: # config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = [] # "hour", "dayofweek", "day", "month", "year"
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = True
num_stack_levels = 1
num_bag_folds = 0
if auto_stack:
    num_stack_levels = None
    num_bag_folds = None

use_tune_data = False
use_test_data = True
tune_and_test_length = 24*30*3 # 3 months from end, this changes the
    ↪ evaluations for only test
holdout_frac = None
use_bag_holdout = False # Enable this if there is a large gap between score_val
    ↪ and score_test in stack models.

sample_weight = 'sample_weight' #None
weight_evaluation = True #False
sample_weight_estimated = 1 # this changes evaluations for test and tune WTF,
    ↪ cant find a fix

run_analysis = False
```

```

[2]: import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings("ignore")

def fix_datetime(X, name):
    # Convert 'date_forecast' to datetime format and replace original column
    ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    # Drop rows where the minute part of the time is not 0
    X = X[X.index.minute == 0].copy()
    return X

def convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_datetime(X_train_observed, "X_train_observed")
    X_train_estimated = fix_datetime(X_train_estimated, "X_train_estimated")
    X_test = fix_datetime(X_test, "X_test")

    # add sample weights, which are 1 for observed and 3 for estimated
    X_train_observed["sample_weight"] = 1
    X_train_estimated["sample_weight"] = sample_weight_estimated
    X_test["sample_weight"] = sample_weight_estimated

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -
        ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
        ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =
        ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
        ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0

```

```

X_train_estimated["is_estimated"] = 1
X_test["is_estimated"] = 1

X_train_estimated.drop(columns=['date_calc'], inplace=True)
X_test.drop(columns=['date_calc'], inplace=True)

y_train['ds'] = pd.to_datetime(y_train['time'])
y_train.drop(columns=['time'], inplace=True)
y_train.sort_values(by='ds', inplace=True)
y_train.set_index('ds', inplace=True)

return X_train_observed, X_train_estimated, X_test, y_train

def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,
↳location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =
↳convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # fill missng sample_weight with 3
    #X_train["sample_weight"] = X_train["sample_weight"].fillna(0)

    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,
↳right_index=True)

    # print number of nans in sample_weight
    print(f"Number of nans in sample_weight: {X_train['sample_weight'].isna().
↳sum()}")

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    X_train["location"] = location
    X_test["location"] = location

```

```

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed = pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,
    ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)

```

```

Processing location A...
Number of nans in sample_weight: 0
Number of nans in y: 0
Processing location B...
Number of nans in sample_weight: 0
Number of nans in y: 4
Processing location C...
Number of nans in sample_weight: 0
Number of nans in y: 6059

```

1 Feature engineering

```
[3]: import numpy as np
import pandas as pd

X_train.dropna(subset=['y'], inplace=True)

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())

if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique() # Assuming 'location' is the name
    ↪ of the column representing locations

    grouped_dfs = [] # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),
    ↪ repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())
```

```
to_drop = ["snow_drift:idx", "snow_density:kgm3"]
```

```
X_train.drop(columns=to_drop, inplace=True)
```

```
X_test.drop(columns=to_drop, inplace=True)
```

```
X_train.to_csv('X_train_raw.csv', index=True)
```

```
X_test.to_csv('X_test_raw.csv', index=True)
```

```

                                absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                                7.7                1.230
2019-06-02 23:00:00                                7.7                1.225
2019-06-03 00:00:00                                7.7                1.221
2019-06-03 01:00:00                                8.2                1.218
2019-06-03 02:00:00                                8.8                1.219

```

```

                                ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00                1744.900024                0.000000
2019-06-02 23:00:00                1703.599976                0.000000
2019-06-03 00:00:00                1668.099976                0.000000
2019-06-03 01:00:00                1388.400024                0.000000
2019-06-03 02:00:00                1108.500000                6546.899902

```

```

                                clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00                0.0                1744.900024                0.0
2019-06-02 23:00:00                0.0                1703.599976                0.0
2019-06-03 00:00:00                0.0                1668.099976                0.0
2019-06-03 01:00:00                0.0                1388.400024                0.0
2019-06-03 02:00:00                9.8                1108.500000                0.0

```

```

                                dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  ...  \
ds
2019-06-02 22:00:00                280.299988                0.0                0.000000  ...
2019-06-02 23:00:00                280.299988                0.0                0.000000  ...
2019-06-03 00:00:00                280.200012                0.0                0.000000  ...
2019-06-03 01:00:00                281.299988                0.0                0.000000  ...
2019-06-03 02:00:00                282.299988                4.3                7743.299805  ...

```

```

                                total_cloud_cover:p  visibility:m  wind_speed_10m:ms  \
ds
2019-06-02 22:00:00                100.0  39640.101562                3.7
2019-06-02 23:00:00                100.0  41699.898438                3.5
2019-06-03 00:00:00                100.0  20473.000000                3.2

```

2019-06-03 01:00:00	100.0	2104.600098	2.8
2019-06-03 02:00:00	100.0	2681.600098	2.7

	wind_speed_u_10m:ms	wind_speed_v_10m:ms	\
ds			
2019-06-02 22:00:00	-3.6	-0.8	
2019-06-02 23:00:00	-3.5	0.0	
2019-06-03 00:00:00	-3.1	0.7	
2019-06-03 01:00:00	-2.7	0.8	
2019-06-03 02:00:00	-2.5	1.0	

	wind_speed_w_1000hPa:ms	sample_weight	is_estimated	\
ds				
2019-06-02 22:00:00	-0.0	1	0	
2019-06-02 23:00:00	-0.0	1	0	
2019-06-03 00:00:00	-0.0	1	0	
2019-06-03 01:00:00	-0.0	1	0	
2019-06-03 02:00:00	-0.0	1	0	

	y	location
ds		
2019-06-02 22:00:00	0.00	A
2019-06-02 23:00:00	0.00	A
2019-06-03 00:00:00	0.00	A
2019-06-03 01:00:00	0.00	A
2019-06-03 02:00:00	19.36	A

[5 rows x 49 columns]

```
[4]: from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
train_data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the
# first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())

# get end date of train data and subtract 3 months
split_time = pd.to_datetime(train_data["ds"]).max() - pd.
    Timedelta(hours=tune_and_test_length)
```

```

train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])

if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])

def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /
        loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            loc_tuning_data = loc_test_set.iloc[:len(loc_test_set)//2]
            loc_test_data = loc_test_set.iloc[len(loc_test_set)//2:]
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
        shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

        # ensure sample weights for your tuning data sum to the number of rows in
        the tuning data.
        tuning_data = normalize_sample_weights_per_location(tuning_data)

else:
    if use_test_data:

```



```

train_data = train_set
test_data = test_set
print("Shape of test", test_data.shape[0])

# ensure sample weights for your training (or tuning) data sum to the number of
rows in the training (or tuning) data.
train_data = normalize_sample_weights_per_location(train_data)
if use_test_data:
    test_data = normalize_sample_weights_per_location(test_data)

```

Shape of test 5791

```

[5]: if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,
label="y", sample=None)

```

```

[6]: if run_analysis:
    auto.target_analysis(train_data=train_data, label="y")

```

2 Starting

```

[7]: import os

# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for
filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)

```

Last submission number: 85
Now creating submission number: 86
New filename: submission_86

```

[8]: predictors = [None, None, None]

```

```

[9]: def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both
    ↪train and tune data and test data
    print("Train data sample weight sum:", train_data[train_data["location"] ==
    ↪loc]["sample_weight"].sum())
    print("Train data number of rows:", train_data[train_data["location"] ==
    ↪loc].shape[0])
    if use_tune_data:
        print("Tune data sample weight sum:",
    ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
        print("Tune data number of rows:", tuning_data[tuning_data["location"]
    ↪== loc].shape[0])
    if use_test_data:
        print("Test data sample weight sum:", test_data[test_data["location"]
    ↪== loc]["sample_weight"].sum())
        print("Test data number of rows:", test_data[test_data["location"] ==
    ↪loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        sample_weight=sample_weight,
        weight_evaluation=weight_evaluation,
        groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc],
        time_limit=time_limit,
        #presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2, # just put
    ↪somethin, will be overwritten anyways
        tuning_data=tuning_data[tuning_data["location"] == loc] if
    ↪use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
    ↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

```

```

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)

```

Training model for location A...

Train data sample weight sum: 31900

Train data number of rows: 31900

Test data sample weight sum: 2161

Test data number of rows: 2161

Values in column 'sample_weight' used as sample weights instead of predictive features. Evaluation will report weighted metrics, so ensure same column exists in test data.

Beginning AutoGluon training ... Time limit = 1800s

AutoGluon will save models to "AutogluonModels/submission_86_A/"

AutoGluon Version: 0.8.2

Python Version: 3.10.12

Operating System: Linux

Platform Machine: x86_64

Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)

Disk Space Avail: 305.89 GB / 315.93 GB (96.8%)

Train Data Rows: 31900

Train Data Columns: 46

Label Column: y

Preprocessing data ...

AutoGluon infers your prediction problem is: 'regression' (because dtype of label-column == float and many unique label-values observed).

Label info (max, min, mean, stddev): (5733.42, 0.0, 633.132, 1165.64686)

If 'regression' is not the correct problem_type, please manually specify the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

Using Feature Generators to preprocess the data ...

Fitting AutoMLPipelineFeatureGenerator...

Available Memory: 132480.39 MB

Train Data (Original) Memory Usage: 13.08 MB (0.0% of available memory)

Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.

Stage 1 Generators:

Fitting AsTypeFeatureGenerator...

Note: Converting 4 features to boolean dtype as they only contain 2 unique values.

Stage 2 Generators:

Fitting FillNaFeatureGenerator...

Stage 3 Generators:

Fitting IdentityFeatureGenerator...

Stage 4 Generators:

```

        Fitting DropUniqueFeatureGenerator...
Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...
Useless Original Features (Count: 2): ['elevation:m', 'location']
        These features carry no predictive signal and should be manually
investigated.

        This is typically a feature which has the same value for all
rows.

        These features do not need to be present at inference time.
Types of features in original data (raw dtype, special dtypes):
        ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', [])   : 1 | ['is_estimated']
Types of features in processed data (raw dtype, special dtypes):
        ('float', []) : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', ['bool']) : 4 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms', 'is_estimated']
0.2s = Fit runtime
43 features in original data used to generate 43 features in processed
data.

Train Data (Processed) Memory Usage: 10.08 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.24s ...
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with
holdout_frac=0.07836990595611286, Train Rows: 29400, Val Rows: 2500
User-specified model hyperparameters to be fit:
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}]},
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},

```

```

{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
  'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif ... Training model for up to 1799.76s of the
1799.76s of remaining time.
    -285.3795          = Validation score    (-mean_absolute_error)
    0.04s           = Training    runtime
    0.08s           = Validation runtime
Fitting model: KNeighborsDist ... Training model for up to 1799.63s of the
1799.63s of remaining time.
    -288.0072          = Validation score    (-mean_absolute_error)
    0.04s           = Training    runtime
    0.04s           = Validation runtime
Fitting model: LightGBMXT ... Training model for up to 1799.55s of the 1799.55s
of remaining time.

[1000]  valid_set's l1: 180.228
[2000]  valid_set's l1: 176.224
[3000]  valid_set's l1: 172.788
[4000]  valid_set's l1: 171.595
[5000]  valid_set's l1: 170.64
[6000]  valid_set's l1: 169.711
[7000]  valid_set's l1: 169.163
[8000]  valid_set's l1: 168.64
[9000]  valid_set's l1: 168.444
[10000] valid_set's l1: 168.216

    -168.1973          = Validation score    (-mean_absolute_error)
    13.53s          = Training    runtime
    0.16s           = Validation runtime
Fitting model: LightGBM ... Training model for up to 1785.57s of the 1785.57s of
remaining time.

[1000]  valid_set's l1: 183.003
[2000]  valid_set's l1: 180.902
[3000]  valid_set's l1: 179.583
[4000]  valid_set's l1: 179.336
[5000]  valid_set's l1: 178.787
[6000]  valid_set's l1: 178.543
[7000]  valid_set's l1: 178.369
[8000]  valid_set's l1: 178.173
[9000]  valid_set's l1: 178.088
[10000] valid_set's l1: 178.079

    -178.0769          = Validation score    (-mean_absolute_error)
    14.02s          = Training    runtime
    0.19s           = Validation runtime

```

Fitting model: RandomForestMSE ... Training model for up to 1770.99s of the 1770.98s of remaining time.

```

-187.4079      = Validation score    (-mean_absolute_error)
7.52s         = Training   runtime
0.09s         = Validation runtime

```

Fitting model: CatBoost ... Training model for up to 1762.91s of the 1762.9s of remaining time.

```

-181.8324      = Validation score    (-mean_absolute_error)
117.12s        = Training   runtime
0.01s          = Validation runtime

```

Fitting model: ExtraTreesMSE ... Training model for up to 1645.73s of the 1645.73s of remaining time.

```

-186.5913      = Validation score    (-mean_absolute_error)
1.7s           = Training   runtime
0.1s           = Validation runtime

```

Fitting model: NeuralNetFastAI ... Training model for up to 1643.46s of the 1643.45s of remaining time.

```

-192.7725      = Validation score    (-mean_absolute_error)
28.01s         = Training   runtime
0.04s          = Validation runtime

```

Fitting model: XGBoost ... Training model for up to 1615.36s of the 1615.35s of remaining time.

```

-189.7843      = Validation score    (-mean_absolute_error)
1.13s          = Training   runtime
0.01s          = Validation runtime

```

Fitting model: NeuralNetTorch ... Training model for up to 1614.21s of the 1614.2s of remaining time.

```

-176.9104      = Validation score    (-mean_absolute_error)
70.49s         = Training   runtime
0.04s          = Validation runtime

```

Fitting model: LightGBMLarge ... Training model for up to 1543.67s of the 1543.67s of remaining time.

```

[1000] valid_set's l1: 170.334
[2000] valid_set's l1: 168.636
[3000] valid_set's l1: 168.404
[4000] valid_set's l1: 168.258
[5000] valid_set's l1: 168.207
[6000] valid_set's l1: 168.19
[7000] valid_set's l1: 168.181
[8000] valid_set's l1: 168.177
[9000] valid_set's l1: 168.175
[10000] valid_set's l1: 168.174

-168.1739      = Validation score    (-mean_absolute_error)
44.22s         = Training   runtime
0.29s          = Validation runtime

```

Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the 1497.99s of remaining time.

```

-161.7021      = Validation score    (-mean_absolute_error)
0.47s         = Training    runtime
0.0s          = Validation runtime
AutoGluon training complete, total runtime = 302.51s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_86_A/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -193.31355744877953
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -193.31355744877953,
    "root_mean_squared_error": -424.88059831295095,
    "mean_squared_error": -180523.5228227712,
    "r2": 0.8690186704532171,
    "pearsonr": 0.9338388267518068,
    "median_absolute_error": -12.229445571899415
}

Evaluation on test data:
-193.31355744877953

```

```

[10]: loc = "B"
      predictors[1] = fit_predictor_for_location(loc)

```

Values in column 'sample_weight' used as sample weights instead of predictive features. Evaluation will report weighted metrics, so ensure same column exists in test data.

Beginning AutoGluon training ... Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_86_B/"

```

AutoGluon Version: 0.8.2
Python Version:    3.10.12
Operating System:  Linux
Platform Machine:  x86_64
Platform Version:  #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:  304.91 GB / 315.93 GB (96.5%)
Train Data Rows:   30768
Train Data Columns: 46

```

Label Column: y

Preprocessing data ...

AutoGluon infers your prediction problem is: 'regression' (because dtype of label-column == float and many unique label-values observed).

Label info (max, min, mean, stddev): (1152.3, -0.0, 97.74541, 195.0957)

If 'regression' is not the correct problem_type, please manually specify the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

```

Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
    Available Memory: 130671.65 MB
    Train Data (Original) Memory Usage: 12.62 MB (0.0% of available memory)
    Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
    Stage 1 Generators:
        Fitting AsTypeFeatureGenerator...
            Note: Converting 4 features to boolean dtype as they
only contain 2 unique values.
    Stage 2 Generators:
        Fitting FillNaFeatureGenerator...
    Stage 3 Generators:
        Fitting IdentityFeatureGenerator...
    Stage 4 Generators:
        Fitting DropUniqueFeatureGenerator...

Training model for location B...
Train data sample weight sum: 30768
Train data number of rows: 30768
Test data sample weight sum: 2051
Test data number of rows: 2051

    Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...
    Useless Original Features (Count: 2): ['elevation:m', 'location']
    These features carry no predictive signal and should be manually
investigated.
        This is typically a feature which has the same value for all
rows.
        These features do not need to be present at inference time.
    Types of features in original data (raw dtype, special dtypes):
        ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', []) : 1 | ['is_estimated']
    Types of features in processed data (raw dtype, special dtypes):
        ('float', []) : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', ['bool']) : 4 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms', 'is_estimated']
    0.2s = Fit runtime
    43 features in original data used to generate 43 features in processed
data.
    Train Data (Processed) Memory Usage: 9.72 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.19s ...
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'

```


This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()

Automatically generating train/validation split with

holdout_frac=0.0812532501300052, Train Rows: 28268, Val Rows: 2500

User-specified model hyperparameters to be fit:

```
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
    'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
```

Fitting 11 L1 models ...

Fitting model: KNeighborsUnif ... Training model for up to 1799.81s of the
1799.8s of remaining time.

-57.0973 = Validation score (-mean_absolute_error)

0.04s = Training runtime

0.04s = Validation runtime

Fitting model: KNeighborsDist ... Training model for up to 1799.72s of the
1799.72s of remaining time.

-56.8968 = Validation score (-mean_absolute_error)

0.04s = Training runtime

0.04s = Validation runtime

Fitting model: LightGBMXT ... Training model for up to 1799.64s of the 1799.63s
of remaining time.

[1000] valid_set's l1: 35.694

[2000] valid_set's l1: 33.3014

[3000] valid_set's l1: 32.221

[4000] valid_set's l1: 31.4655

[5000] valid_set's l1: 30.9341

[6000] valid_set's l1: 30.56

[7000] valid_set's l1: 30.2521

[8000] valid_set's l1: 29.991

```

[9000] valid_set's l1: 29.7907
[10000] valid_set's l1: 29.6708

-29.6708          = Validation score    (-mean_absolute_error)
17.43s           = Training    runtime
0.18s            = Validation runtime
Fitting model: LightGBM ... Training model for up to 1781.67s of the 1781.67s of
remaining time.

[1000] valid_set's l1: 33.1181
[2000] valid_set's l1: 31.7424
[3000] valid_set's l1: 30.9712
[4000] valid_set's l1: 30.5924
[5000] valid_set's l1: 30.423
[6000] valid_set's l1: 30.3091
[7000] valid_set's l1: 30.2459
[8000] valid_set's l1: 30.2123
[9000] valid_set's l1: 30.1696
[10000] valid_set's l1: 30.1353

-30.1351          = Validation score    (-mean_absolute_error)
13.95s           = Training    runtime
0.16s            = Validation runtime
Fitting model: RandomForestMSE ... Training model for up to 1767.31s of the
1767.31s of remaining time.

-35.3213          = Validation score    (-mean_absolute_error)
8.69s            = Training    runtime
0.09s            = Validation runtime
Fitting model: CatBoost ... Training model for up to 1758.19s of the 1758.18s of
remaining time.

-32.5119          = Validation score    (-mean_absolute_error)
120.77s          = Training    runtime
0.01s            = Validation runtime
Fitting model: ExtraTreesMSE ... Training model for up to 1637.38s of the
1637.37s of remaining time.

-36.3782          = Validation score    (-mean_absolute_error)
1.84s            = Training    runtime
0.1s             = Validation runtime
Fitting model: NeuralNetFastAI ... Training model for up to 1635.03s of the
1635.02s of remaining time.

-39.8115          = Validation score    (-mean_absolute_error)
26.63s           = Training    runtime
0.04s            = Validation runtime
Fitting model: XGBoost ... Training model for up to 1608.32s of the 1608.32s of
remaining time.

-33.0851          = Validation score    (-mean_absolute_error)
24.77s           = Training    runtime
0.21s            = Validation runtime
Fitting model: NeuralNetTorch ... Training model for up to 1583.18s of the

```

```

1583.18s of remaining time.
    -33.8046          = Validation score    (-mean_absolute_error)
    123.74s   = Training   runtime
    0.04s     = Validation runtime
Fitting model: LightGBMLarge ... Training model for up to 1459.39s of the
1459.39s of remaining time.

[1000] valid_set's l1: 30.2138
[2000] valid_set's l1: 29.1566
[3000] valid_set's l1: 28.9192
[4000] valid_set's l1: 28.8311
[5000] valid_set's l1: 28.7995
[6000] valid_set's l1: 28.7854
[7000] valid_set's l1: 28.7794
[8000] valid_set's l1: 28.7772
[9000] valid_set's l1: 28.7759
[10000] valid_set's l1: 28.7755

    -28.7755          = Validation score    (-mean_absolute_error)
    42.98s   = Training   runtime
    0.31s     = Validation runtime
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
1414.94s of remaining time.
    -28.2699          = Validation score    (-mean_absolute_error)
    0.46s   = Training   runtime
    0.0s     = Validation runtime
AutoGluon training complete, total runtime = 385.56s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_86_B/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -36.68013864727576
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -36.68013864727576,
    "root_mean_squared_error": -80.47978000367722,
    "mean_squared_error": -6476.994989440284,
    "r2": 0.7916806334883096,
    "pearsonr": 0.9099906161614322,
    "median_absolute_error": -8.040388107299805
}

Evaluation on test data:
-36.68013864727576

```

```
[11]: loc = "C"  
      predictors[2] = fit_predictor_for_location(loc)
```

Values in column 'sample_weight' used as sample weights instead of predictive features. Evaluation will report weighted metrics, so ensure same column exists in test data.

Beginning AutoGluon training ... Time limit = 1800s

AutoGluon will save models to "AutogluonModels/submission_86_C/"

AutoGluon Version: 0.8.2

Python Version: 3.10.12

Operating System: Linux

Platform Machine: x86_64

Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)

Disk Space Avail: 304.00 GB / 315.93 GB (96.2%)

Train Data Rows: 24492

Train Data Columns: 46

Label Column: y

Preprocessing data ...

AutoGluon infers your prediction problem is: 'regression' (because dtype of label-column == float and label-values can't be converted to int).

Label info (max, min, mean, stddev): (999.6, 0.0, 78.11911, 167.50151)

If 'regression' is not the correct problem_type, please manually specify the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

Using Feature Generators to preprocess the data ...

Fitting AutoMLPipelineFeatureGenerator...

Available Memory: 130462.48 MB

Train Data (Original) Memory Usage: 10.04 MB (0.0% of available memory)

Inferring data type of each feature based on column values. Set

feature_metadata_in to manually specify special dtypes of the features.

Stage 1 Generators:

Fitting AsTypeFeatureGenerator...

Note: Converting 3 features to boolean dtype as they only contain 2 unique values.

Stage 2 Generators:

Fitting FillNaFeatureGenerator...

Stage 3 Generators:

Fitting IdentityFeatureGenerator...

Stage 4 Generators:

Fitting DropUniqueFeatureGenerator...

Stage 5 Generators:

Fitting DropDuplicatesFeatureGenerator...

Useless Original Features (Count: 2): ['elevation:m', 'location']

These features carry no predictive signal and should be manually investigated.

Training model for location C...

Train data sample weight sum: 24492

Train data number of rows: 24492
Test data sample weight sum: 1579
Test data number of rows: 1579

This is typically a feature which has the same value for all rows.

These features do not need to be present at inference time.

Types of features in original data (raw dtype, special dtypes):

```
('float', []) : 42 | ['absolute_humidity_2m:gm3',  
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',  
'clear_sky_rad:W', ...]
```

```
('int', []) : 1 | ['is_estimated']
```

Types of features in processed data (raw dtype, special dtypes):

```
('float', []) : 40 | ['absolute_humidity_2m:gm3',  
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',  
'clear_sky_rad:W', ...]
```

```
('int', ['bool']) : 3 | ['is_day:idx', 'is_in_shadow:idx',  
'is_estimated']
```

0.1s = Fit runtime

43 features in original data used to generate 43 features in processed data.

Train Data (Processed) Memory Usage: 7.91 MB (0.0% of available memory)

Data preprocessing and feature engineering runtime = 0.17s ...

AutoGluon will gauge predictive performance using evaluation metric:

'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()

Automatically generating train/validation split with holdout_frac=0.1, Train Rows: 22042, Val Rows: 2450

User-specified model hyperparameters to be fit:

```
{  
    'NN_TORCH': {},  
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],  
'GBMLarge'],  
    'CAT': {},  
    'XGB': {},  
    'FASTAI': {},  
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',  
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':  
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},  
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',  
'problem_types': ['regression', 'quantile']}}],  
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',  
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':  
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},  
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',  
'problem_types': ['regression', 'quantile']}}],  
}
```

```

    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif ... Training model for up to 1799.83s of the
1799.83s of remaining time.
    -33.2822          = Validation score    (-mean_absolute_error)
    0.03s           = Training    runtime
    0.03s           = Validation runtime
Fitting model: KNeighborsDist ... Training model for up to 1799.76s of the
1799.76s of remaining time.
    -33.3446          = Validation score    (-mean_absolute_error)
    0.03s           = Training    runtime
    0.03s           = Validation runtime
Fitting model: LightGBMXT ... Training model for up to 1799.68s of the 1799.68s
of remaining time.

[1000]  valid_set's l1: 18.9075
[2000]  valid_set's l1: 18.4144
[3000]  valid_set's l1: 18.1066
[4000]  valid_set's l1: 17.943
[5000]  valid_set's l1: 17.8413
[6000]  valid_set's l1: 17.8265
[7000]  valid_set's l1: 17.7906
[8000]  valid_set's l1: 17.7815
[9000]  valid_set's l1: 17.7635
[10000] valid_set's l1: 17.7465

    -17.7423          = Validation score    (-mean_absolute_error)
    12.24s           = Training    runtime
    0.16s           = Validation runtime
Fitting model: LightGBM ... Training model for up to 1787.04s of the 1787.04s of
remaining time.

[1000]  valid_set's l1: 19.0285
[2000]  valid_set's l1: 18.7978
[3000]  valid_set's l1: 18.7218
[4000]  valid_set's l1: 18.6801
[5000]  valid_set's l1: 18.6601
[6000]  valid_set's l1: 18.6534
[7000]  valid_set's l1: 18.6481
[8000]  valid_set's l1: 18.646
[9000]  valid_set's l1: 18.6472
[10000] valid_set's l1: 18.6465

    -18.6454          = Validation score    (-mean_absolute_error)
    12.59s           = Training    runtime
    0.14s           = Validation runtime
Fitting model: RandomForestMSE ... Training model for up to 1774.12s of the
1774.12s of remaining time.

```

```

-20.2132          = Validation score    (-mean_absolute_error)
4.65s            = Training    runtime
0.08s            = Validation runtime
Fitting model: CatBoost ... Training model for up to 1769.22s of the
remaining time.
-18.5986          = Validation score    (-mean_absolute_error)
119.73s          = Training    runtime
0.01s            = Validation runtime
Fitting model: ExtraTreesMSE ... Training model for up to 1649.44s of the
1649.44s of remaining time.
-20.2361          = Validation score    (-mean_absolute_error)
1.04s            = Training    runtime
0.08s            = Validation runtime
Fitting model: NeuralNetFastAI ... Training model for up to 1648.14s of the
1648.14s of remaining time.
-20.7285          = Validation score    (-mean_absolute_error)
21.81s           = Training    runtime
0.04s            = Validation runtime
Fitting model: XGBoost ... Training model for up to 1626.26s of the 1626.26s of
remaining time.
-19.187           = Validation score    (-mean_absolute_error)
24.04s           = Training    runtime
0.21s            = Validation runtime
Fitting model: NeuralNetTorch ... Training model for up to 1601.86s of the
1601.86s of remaining time.
-19.6345          = Validation score    (-mean_absolute_error)
32.72s           = Training    runtime
0.03s            = Validation runtime
Fitting model: LightGBMLarge ... Training model for up to 1569.1s of the 1569.1s
of remaining time.

[1000]  valid_set's l1: 18.2934
[2000]  valid_set's l1: 18.1615
[3000]  valid_set's l1: 18.1423
[4000]  valid_set's l1: 18.1367
[5000]  valid_set's l1: 18.1351
[6000]  valid_set's l1: 18.1345
[7000]  valid_set's l1: 18.1343
[8000]  valid_set's l1: 18.1342
[9000]  valid_set's l1: 18.1342
[10000] valid_set's l1: 18.1342

-18.1342          = Validation score    (-mean_absolute_error)
45.54s           = Training    runtime
0.34s            = Validation runtime
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
1522.06s of remaining time.
-17.2489          = Validation score    (-mean_absolute_error)
0.45s            = Training    runtime

```

```

0.0s      = Validation runtime
AutoGluon training complete, total runtime = 278.42s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_86_C/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -32.53914948780841
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -32.53914948780841,
    "root_mean_squared_error": -66.06150412414301,
    "mean_squared_error": -4364.122327144165,
    "r2": 0.7711882838282728,
    "pearsonr": 0.8967034153460325,
    "median_absolute_error": -3.294370412826538
}

Evaluation on test data:
-32.53914948780841

```

3 Submit

```

[12]: import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data

```

```

Loaded data from: X_train_raw.csv | Columns = 48 / 48 | Rows = 92951 -> 92951
Loaded data from: X_test_raw.csv | Columns = 47 / 47 | Rows = 2160 -> 2160

```

```

[13]: test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",
↪ "location"], left_on=["ds", "location"])

#test_data_merged

```

```

Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160

```



```

[14]: # predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
    ↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
    ↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,
    ↪"prediction"] = past_pred

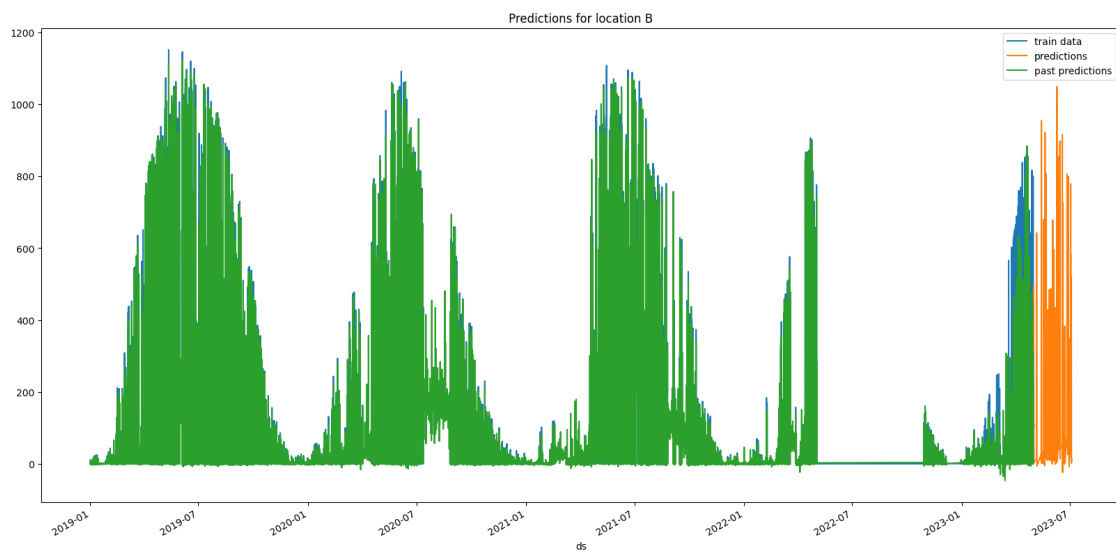
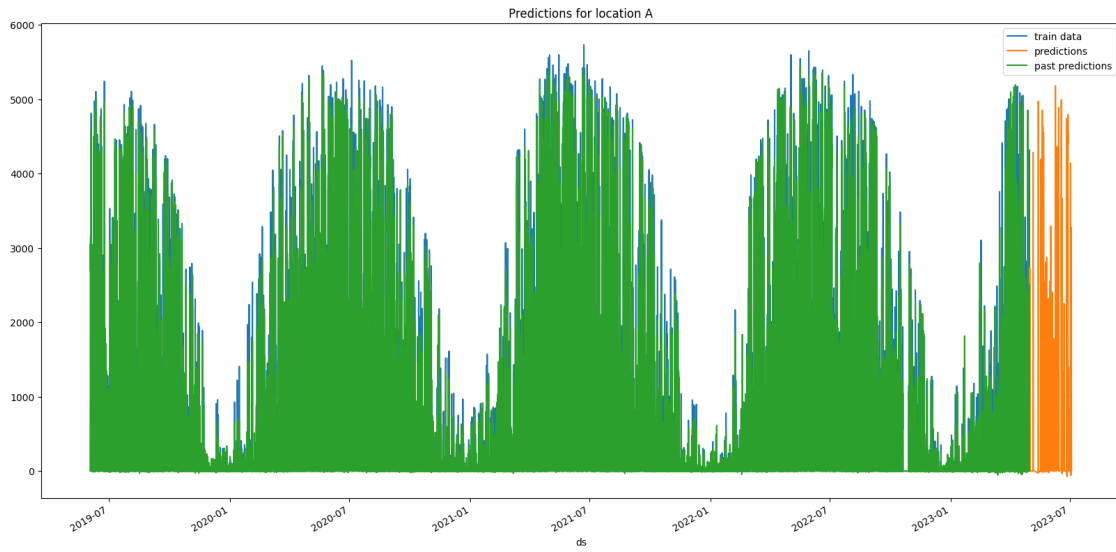
[15]: # plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='y', ax=ax, label="train data")

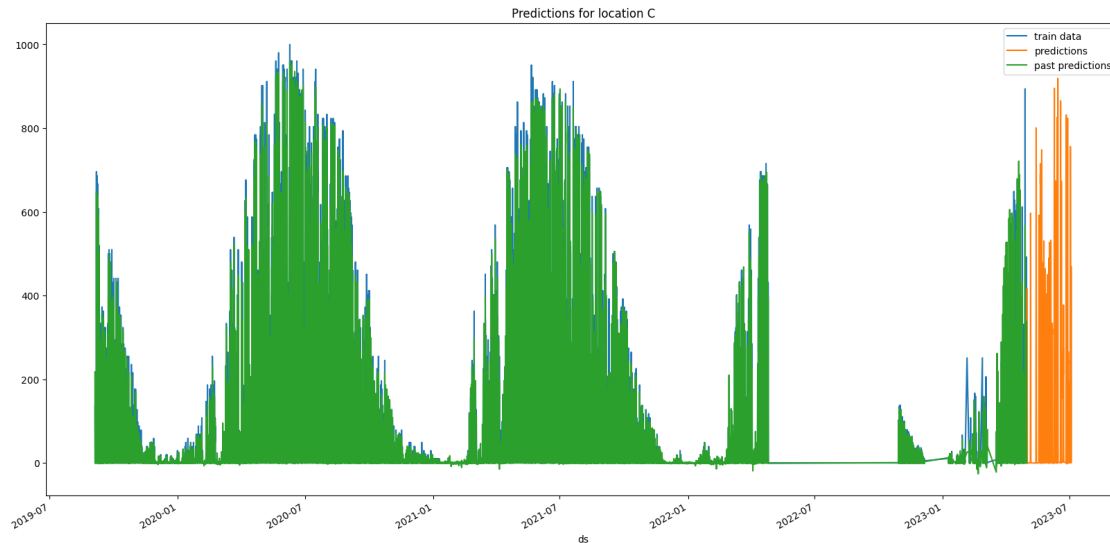
    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")

```





```
[16]: # concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
[16]:      id  prediction
0      0   -2.138088
1      1   -2.369956
2      2    2.265450
3      3   35.582993
4      4  420.204895
..    ...      ...
715  2155   51.505508
716  2156   30.130787
717  2157   13.440950
718  2158    2.201931
719  2159    1.701847
```

[2160 rows x 2 columns]

```
[17]: # Save the submission DataFrame to submissions folder, create new name based on
      ↳ last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
      ↳ index=False)
print("jallia")
```

Saving submission to submissions/submission_86.csv
jall1a

```
[18]: # save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

<IPython.core.display.Javascript object>

```
[19]: # save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
    ↪join('notebook_pdfs', f'{new_filename}.pdf'), "autogluon_each_location.
    ↪ipynb"])
```

[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to <https://pandoc.org/installing.html>.
Continuing with doubts...
check_pandoc_version()
[NbConvertApp] Writing 113058 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 90560 bytes to notebook_pdfs/submission_86.pdf

```
[19]: CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
    'notebook_pdfs/submission_86.pdf', 'autogluon_each_location.ipynb'],
    returncode=0)
```

```
[20]: # feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,
    ↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using 4394 rows with 10 shuffle sets... Time limit: 600s...

595.97s = Expected runtime (59.6s per shuffle set)

338.5s = Actual runtime (Completed 10 of 10 shuffle sets)

[20]:	importance	stddev	p_value	n \
direct_rad:W	1.456169e+02	2.106609	2.232835e-18	10
clear_sky_rad:W	7.906986e+01	1.975863	3.051799e-16	10
diffuse_rad:W	7.745410e+01	2.498507	3.033211e-15	10
sun_azimuth:d	5.803144e+01	3.402080	6.502557e-13	10
sun_elevation:d	3.151281e+01	0.936457	1.450224e-15	10
clear_sky_energy_1h:J	2.163643e+01	1.374638	1.337898e-12	10
direct_rad_1h:J	2.082088e+01	0.584796	8.732321e-16	10
total_cloud_cover:p	1.775258e+01	1.033124	6.087668e-13	10
effective_cloud_cover:p	1.230926e+01	0.960829	8.469377e-12	10
diffuse_rad_1h:J	1.219040e+01	1.296686	1.346458e-10	10
wind_speed_u_10m:ms	9.109273e+00	0.976982	1.449326e-10	10
is_day:idx	6.694744e+00	0.297911	5.468339e-14	10
cloud_base_agl:m	6.512255e+00	0.602472	3.873097e-11	10
visibility:m	5.528293e+00	0.672025	4.420931e-10	10
fresh_snow_24h:cm	5.277689e+00	0.854260	5.581422e-09	10
is_in_shadow:idx	5.182462e+00	0.601572	2.932146e-10	10
relative_humidity_1000hPa:p	4.702148e+00	0.590485	5.902173e-10	10
ceiling_height_agl:m	4.033180e+00	0.585065	2.120870e-09	10
msl_pressure:hPa	3.952032e+00	0.825533	5.207290e-08	10
wind_speed_v_10m:ms	3.912216e+00	0.578978	2.531084e-09	10
snow_water:kgm2	3.868391e+00	0.744507	2.550402e-08	10
wind_speed_10m:ms	3.711112e+00	0.603929	5.852194e-09	10
pressure_100m:hPa	2.391802e+00	0.501463	5.376686e-08	10
dew_point_2m:K	1.785512e+00	0.479794	4.545219e-07	10
pressure_50m:hPa	1.613141e+00	0.380217	1.480362e-07	10
fresh_snow_12h:cm	1.567123e+00	0.581924	6.694266e-06	10
fresh_snow_6h:cm	1.455185e+00	0.526195	5.396138e-06	10
t_1000hPa:K	1.331368e+00	1.397898	7.336768e-03	10
sfc_pressure:hPa	1.091553e+00	0.527098	5.266605e-05	10
super_cooled_liquid_water:kgm2	6.478008e-01	0.600171	3.854995e-03	10
snow_depth:cm	6.143269e-01	0.254087	1.586406e-05	10
precip_type_5min:idx	5.475742e-01	0.594144	8.593630e-03	10
precip_5min:mm	5.436858e-01	0.441514	1.826117e-03	10
fresh_snow_1h:cm	4.210567e-01	0.216952	8.566489e-05	10
dew_or_rime:idx	3.345875e-01	0.211474	3.678404e-04	10
prob_rime:p	3.264448e-01	0.207029	3.764629e-04	10
air_density_2m:kgm3	2.561204e-01	0.757439	1.563883e-01	10
rain_water:kgm2	5.075184e-02	0.225176	2.470352e-01	10
wind_speed_w_1000hPa:ms	5.556227e-08	0.000000	5.000000e-01	10

is_estimated	0.000000e+00	0.000000	5.000000e-01	10
fresh_snow_3h:cm	-8.933769e-03	0.334399	5.327391e-01	10
absolute_humidity_2m:gm3	-6.031311e-02	0.255337	7.629258e-01	10
snow_melt_10min:mm	-2.576696e-01	0.139641	9.998758e-01	10

	p99_high	p99_low
direct_rad:W	1.477818e+02	1.434519e+02
clear_sky_rad:W	8.110043e+01	7.703929e+01
diffuse_rad:W	8.002178e+01	7.488641e+01
sun_azimuth:d	6.152772e+01	5.453516e+01
sun_elevation:d	3.247520e+01	3.055043e+01
clear_sky_energy_1h:J	2.304913e+01	2.022373e+01
direct_rad_1h:J	2.142187e+01	2.021989e+01
total_cloud_cover:p	1.881431e+01	1.669085e+01
effective_cloud_cover:p	1.329669e+01	1.132183e+01
diffuse_rad_1h:J	1.352299e+01	1.085781e+01
wind_speed_u_10m:ms	1.011331e+01	8.105240e+00
is_day:idx	7.000904e+00	6.388584e+00
cloud_base_agl:m	7.131409e+00	5.893102e+00
visibility:m	6.218925e+00	4.837661e+00
fresh_snow_24h:cm	6.155602e+00	4.399776e+00
is_in_shadow:idx	5.800690e+00	4.564234e+00
relative_humidity_1000hPa:p	5.308983e+00	4.095314e+00
ceiling_height_agl:m	4.634445e+00	3.431915e+00
msl_pressure:hPa	4.800423e+00	3.103642e+00
wind_speed_v_10m:ms	4.507225e+00	3.317208e+00
snow_water:kgm2	4.633512e+00	3.103271e+00
wind_speed_10m:ms	4.331762e+00	3.090461e+00
pressure_100m:hPa	2.907150e+00	1.876454e+00
dew_point_2m:K	2.278591e+00	1.292434e+00
pressure_50m:hPa	2.003886e+00	1.222396e+00
fresh_snow_12h:cm	2.165160e+00	9.690859e-01
fresh_snow_6h:cm	1.995950e+00	9.144203e-01
t_1000hPa:K	2.767971e+00	-1.052348e-01
sfc_pressure:hPa	1.633245e+00	5.498607e-01
super_cooled_liquid_water:kgm2	1.264589e+00	3.101234e-02
snow_depth:cm	8.754493e-01	3.532044e-01
precip_type_5min:idx	1.158169e+00	-6.302006e-02
precip_5min:mm	9.974241e-01	8.994744e-02
fresh_snow_1h:cm	6.440157e-01	1.980977e-01
dew_or_rime:idx	5.519163e-01	1.172586e-01
prob_rime:p	5.392061e-01	1.136834e-01
air_density_2m:kgm3	1.034532e+00	-5.222908e-01
rain_water:kgm2	2.821621e-01	-1.806585e-01
wind_speed_w_1000hPa:ms	5.556227e-08	5.556227e-08
is_estimated	0.000000e+00	0.000000e+00
fresh_snow_3h:cm	3.347246e-01	-3.525922e-01

```
absolute_humidity_2m:gm3      2.020938e-01 -3.227201e-01
snow_melt_10min:mm           -1.141624e-01 -4.011769e-01
```

```
[ ]: # feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,
    ↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using 5000 rows with 10 shuffle sets... Time limit: 600s..
634.59s = Expected runtime (63.46s per shuffle set)

```
[ ]: display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
    ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),
    ↪"autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
            ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
            ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',
    ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is
    ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"
```

```

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',
↳ 'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',
↳ 'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])

```